下载APP

06 | Torchvision（上）：数据读取，训练开始的第一步

2021-10-22 方远

《PyTorch深度学习实战》 课程介绍 >

讲述：方远
时长 13:13 大小 12.11M

你好，我是方远。

今天起我们进入模型训练篇的学习。如果将模型看作一辆汽车，那么它的开发过程就可以看作是一套完整的生产流程，环环相扣、缺一不可。这些环节包括**数据的读取、网络的设计、优化方法与损失函数的选择以及一些辅助的工具等**。未来你将尝试构建自己的豪华汽车，或者站在巨人的肩膀上对前人的作品进行优化。

试想一下，如果你对这些基础环节所使用的方法都不清楚，你还能很好地进行下去吗？所以通过这个模块，我们的目标是先把基础打好。通过这模块的学习，对于 PyTorch 都为我们提供了哪些丰富的 API，你就会了然于胸了。

Torchvision 是一个和 PyTorch 配合使用的 Python 包，包含很多图像处理的工具。我们先从数据处理入手，开始 PyTorch 的学习的第一步。这节课我们会先介绍 Torchvision 的常用数据集及其读取方法，在后面的两节课里，我再带你了解常用的图像处理方法与 Torchvision 其它有趣的功能。

PyTorch 中的数据读取

训练开始的第一步，首先就是数据读取。PyTorch 为我们提供了一种十分方便的数据读取机制，即使用 Dataset 类与 DataLoader 类的组合，来得到数据迭代器。在训练或预测时，数据迭代器能够输出每一批次所需的数据，并且对数据进行相应的预处理与数据增强操作。

下面我们分别来看下 Dataset 类与 DataLoader 类。

Dataset 类

PyTorch 中的 Dataset 类是一个抽象类，它可以用来表示数据集。我们通过继承 Dataset 类来自定义数据集的格式、大小和其它属性，后面就可以供 DataLoader 类直接使用。

其实这就表示，无论使用自定义的数据集，还是官方为我们封装好的数据集，其本质都是继承了 Dataset 类。而在继承 Dataset 类时，至少需要重写以下几个方法：

__init__()：构造函数，可自定义数据读取方法以及进行数据预处理；

__len__()：返回数据集大小；

__getitem__()：索引数据集中的某一个数据。

光看原理不容易理解，下面我们来编写一个简单的例子，看下如何使用 Dataset 类定义一个 Tensor 类型的数据集。

复制代码

```
import torch
from torch.utils.data import Dataset

class MyDataset(Dataset):
    # 构造函数
    def __init__(self, data_tensor, target_tensor):
        self.data_tensor = data_tensor
```

```
        self.target_tensor = target_tensor
    # 返回数据集大小
    def __len__(self):
        return self.data_tensor.size(0)
    # 返回索引的数据与标签
    def __getitem__(self, index):
        return self.data_tensor[index], self.target_tensor[index]
```

结合代码可以看到，我们定义了一个名字为 MyDataset 的数据集，在构造函数中，传入 Tensor 类型的数据与标签；在 __len__ 函数中，直接返回 Tensor 的大小；在 __getitem__ 函数中返回索引的数据与标签。

下面，我们来看一下如何调用刚才定义的数据集。首先随机生成一个 10*3 维的数据 Tensor，然后生成 10 维的标签 Tensor，与数据 Tensor 相对应。利用这两个 Tensor，生成一个 MyDataset 的对象。查看数据集的大小可以直接用 len() 函数，索引调用数据可以直接使用下标。

复制代码

```
# 生成数据
data_tensor = torch.randn(10, 3)
target_tensor = torch.randint(2, (10,)) # 标签是0或1

# 将数据封装成Dataset
my_dataset = MyDataset(data_tensor, target_tensor)

# 查看数据集大小
print('Dataset size:', len(my_dataset))
'''
输出：
Dataset size: 10
'''

# 使用索引调用数据
print('tensor_data[0]: ', my_dataset[0])
'''
输出:
tensor_data[0]:  (tensor([ 0.4931, -0.0697,  0.4171]), tensor(0))
'''
```

DataLoader 类

在实际项目中，如果数据量很大，考虑到内存有限、I/O 速度等问题，在训练过程中不可能一次性的将所有数据全部加载到内存中，也不能只用一个进程去加载，所以就需要多进

程、迭代加载，而 DataLoader 就是基于这些需要被设计出来的。

DataLoader 是一个迭代器，最基本的使用方法就是传入一个 Dataset 对象，它会根据参数 batch_size 的值生成一个 batch 的数据，节省内存的同时，它还可以实现多进程、数据打乱等处理。

DataLoader 类的调用方式如下：

复制代码

```
from torch.utils.data import DataLoader
tensor_dataloader = DataLoader(dataset=my_dataset, # 传入的数据集，必须参数
                               batch_size=2,       # 输出的batch大小
                               shuffle=True,       # 数据是否打乱
                               num_workers=0)      # 进程数，0表示只有主进程

# 以循环形式输出
for data, target in tensor_dataloader:
    print(data, target)
'''
输出:
tensor([[-0.1781, -1.1019, -0.1507],
        [-0.6170,  0.2366,  0.1006]]) tensor([0, 0])
tensor([[ 0.9451, -0.4923, -1.8178],
        [-0.4046, -0.5436, -1.7911]]) tensor([0, 0])
tensor([[-0.4561, -1.2480, -0.3051],
        [-0.9738,  0.9465,  0.4812]]) tensor([1, 0])
tensor([[ 0.0260,  1.5276,  0.1687],
        [ 1.3692, -0.0170, -1.6831]]) tensor([1, 0])
tensor([[ 0.0515, -0.8892, -0.1699],
        [ 0.4931, -0.0697,  0.4171]]) tensor([1, 0])
'''

# 输出一个batch
print('One batch tensor data: ', iter(tensor_dataloader).next())
'''
输出:
One batch tensor data:  [tensor([[ 0.9451, -0.4923, -1.8178],
        [-0.4046, -0.5436, -1.7911]]), tensor([0, 0])]
'''
```

结合代码，我们梳理一下 DataLoader 中的几个参数，它们分别表示：

dataset：Dataset 类型，输入的数据集，必须参数；

batch_size：int 类型，每个 batch 有多少个样本；

shuffle：bool 类型，在每个 epoch 开始的时候，是否对数据进行重新打乱；

num_workers：int 类型，加载数据的进程数，0 意味着所有的数据都会被加载进主进程，默认为 0。

什么是 Torchvision

PyTroch 官方为我们提供了一些常用的图片数据集，如果你需要读取这些数据集，那么无需自己实现，只需要利用 Torchvision 就可以搞定。

Torchvision 是一个和 PyTorch 配合使用的 Python 包。它不只提供了一些常用数据集，还提供了几个已经搭建好的经典网络模型，以及集成了一些图像数据处理方面的工具，主要供数据预处理阶段使用。简单地说，Torchvision 库就是**常用数据集 + 常见网络模型 + 常用图像处理方法**。

Torchvision 的安装方式同样非常简单，可以使用 conda 安装，命令如下：

复制代码

```
conda install torchvision -c pytorch
```

或使用 pip 进行安装，命令如下：

复制代码

```
pip install torchvision
```

Torchvision 中默认使用的图像加载器是 PIL，因此为了确保 Torchvision 正常运行，我们还需要安装一个 Python 的第三方图像处理库——Pillow 库。Pillow 提供了广泛的文件格式支持，强大的图像处理能力，主要包括图像储存、图像显示、格式转换以及基本的图像处理操作等。

使用 conda 安装 Pillow 的命令如下：

复制代码

```
conda install pillow
```

使用 pip 安装 Pillow 的命令如下：

复制代码

```
pip install pillow
```

利用 Torchvision 读取数据

安装好 Torchvision 之后，我们再来接着看看。Torchvision 库为我们读取数据提供了哪些支持。

Torchvision 库中的`torchvision.datasets`包中提供了丰富的图像数据集的接口。常用的图像数据集，例如 MNIST、COCO 等，这个模块都为我们做了相应的封装。

下表中列出了`torchvision.datasets`包所有支持的数据集。各个数据集的说明与接口，详见链接 https://pytorch.org/vision/stable/datasets.html。

数据集一览表				
Caltech	FakeData	KITTI	Places365	SVHN
CelebA	Fashion-MNIST	KMNIST	QMNIST	UCF101
CIFAR	Flickr	LSUN	SBD	USPS
Cityscapes	HMDB51	MNIST	SBU	VOC
COCO	ImageNet	Omniglot	SEMEION	WIDERFace
EMNIST	Kinetics-400	PhotoTour	STL10	——

极客时间

这里我想提醒你注意，`torchvision.datasets`这个包本身并不包含数据集的文件本身，它的工作方式是先从网络上把数据集下载到用户指定目录，然后再用它的加载器把数据集加载到内存中。最后，把这个加载后的数据集作为对象返回给用户。

为了让你进一步加深对知识的理解，我们以 MNIST 数据集为例，来说明一下这个模块具体的使用方法。

MNIST 数据集简介

MNIST 数据集是一个著名的手写数字数据集，因为上手简单，在深度学习领域，手写数字识别是一个很经典的学习入门样例。

MNIST 数据集是 NIST 数据集的一个子集，MNIST 数据集你可以通过这里下载。它包含了四个部分，我用表格的方式为你做了梳理。

内容	文件名	其它信息
训练集图片	train-images-idx3-ubyte.gz	9.9 MB，解压后 47 MB，包含6万个样本
训练集标签	train-labels-idx1-ubyte.gz	29 KB，解压后 60 KB，包含6万个标签
测试集图片	t10k-images-idx3-ubyte.gz	1.6 MB，解压后 7.8 MB，包含1万个样本
测试集标签	t10k-labels-idx1-ubyte.gz	5KB，解压后 10 KB，包含1万个标签

极客时间

MNIST 数据集是 ubyte 格式存储，我们先将“训练集图片”解析成图片格式，来直观地看一看数据集具体是什么样子的。具体怎么解析，我在后面数据预览再展开。

极客时间

数据读取

接下来，我们看一下如何使用 Torchvision 来读取 MNIST 数据集。

对于`torchvision.datasets`所支持的所有数据集，它都内置了相应的数据集接口。例如刚才介绍的 MNIST 数据集，`torchvision.datasets`就有一个 MNIST 的接口，接口内封装了从下载、解压缩、读取数据、解析数据等全部过程。

这些接口的工作方式差不多，都是先从网络上把数据集下载到指定目录，然后再用加载器把数据集加载到内存中，最后将加载后的数据集作为对象返回给用户。

以 MNIST 为例，我们可以用如下方式调用：

复制代码

```
# 以MNIST为例
import torchvision
mnist_dataset = torchvision.datasets.MNIST(root='./data',
                                           train=True,
                                           transform=None,
                                           target_transform=None,
                                           download=True)
```

`torchvision.datasets.MNIST`是一个类，对它进行实例化，即可返回一个 MNIST 数据集对象。构造函数包括包含 5 个参数：

root：是一个字符串，用于指定你想要保存 MNIST 数据集的位置。如果 download 是 Flase，则会从目标位置读取数据集；

download：是布尔类型，表示是否下载数据集。如果为 True，则会自动从网上下载这个数据集，存储到 root 指定的位置。如果指定位置已经存在数据集文件，则不会重复下载；

train：是布尔类型，表示是否加载训练集数据。如果为 True，则只加载训练数据。如果为 False，则只加载测试数据集。**这里需要注意，并不是所有的数据集都做了训练集和测试集的划分，这个参数并不一定是有效参数，具体需要参考官方接口说明文档**；

transform：用于对图像进行预处理操作，例如数据增强、归一化、旋转或缩放等。这些操作我们会在下节课展开讲解；

target_transform：用于对图像标签进行预处理操作。

运行上述的代码，我们可以得到下图所示的效果。从图中我们可以看出，程序首先去指定的网址下载了 MNIST 数据集，然后进行了解压缩等操作。如果你再次运行相同的代码，则不会再有下载的过程。

```
# 以MNIST为例
import torchvision
mnist_dataset = torchvision.datasets.MNIST(root='./data',
                                           train=True,
                                           transform=None,
                                           target_transform=None,
                                           download=True)
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to ./data/MNIST/raw/train-images-idx3-ubyte.g
z

9913344/? [23:34<00:00, 7008.43it/s]


Extracting ./data/MNIST/raw/train-images-idx3-ubyte.gz to ./data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz to ./data/MNIST/raw/train-labels-idx1-ubyte.g
z

29696/? [00:06<00:00, 4405.00it/s]


Extracting ./data/MNIST/raw/train-labels-idx1-ubyte.gz to ./data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz to ./data/MNIST/raw/t10k-images-idx3-ubyte.gz

1649664/? [00:05<00:00, 295791.98it/s]


Extracting ./data/MNIST/raw/t10k-images-idx3-ubyte.gz to ./data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz to ./data/MNIST/raw/t10k-labels-idx1-ubyte.gz

5120/? [22:42<00:00, 3.76it/s]


Extracting ./data/MNIST/raw/t10k-labels-idx1-ubyte.gz to ./data/MNIST/raw
```

看到这，你可能还有疑问，好奇我们得到的 mnist_dataset 是什么呢？

如果你用 type 函数查看一下 mnist_dataset 的类型，就可以得到 `torchvision.datasets.mnist.MNIST` ，而这个类是之前我们介绍过的 Dataset 类的派生类。相当于`torchvision.datasets` ，它已经帮我们写好了对 Dataset 类的继承，完成了对数据集的封装，我们直接使用即可。

这里我们主要以 MNIST 为例，进行了说明。其它的数据集使用方法类似，调用的时候你只要需要将类名“MNIST”换成其它数据集名字即可。

对于不同的数据集，数据格式都不尽相同，而`torchvision.datasets`则帮助我们完成了各种不同格式的数据的解析与读取，可以说十分便捷。而对于那些没有官方接口的图像数据集，我们也可以使用以`torchvision.datasets.ImageFolder`接口来自行定义，在图像分类的实战篇中，就是使用 ImageFolder 进行数据读取的，你可以到那个时候再看一看。

数据预览

完成了数据读取工作，我们得到的是对应的 mnist_dataset，刚才已经讲过了，这是一个封装了的数据集。

如果想要查看 mnist_dataset 中的具体内容，我们需要把它转化为列表。（ 如果 IOPub data rate 超限，可以只加载测试集数据，令 train=False ）

复制代码

```
mnist_dataset_list = list(mnist_dataset)
print(mnist_dataset_list)
```

执行结果如下图所示。

```
mnist_dataset_list = list(mnist_dataset)
print(mnist_dataset_list)
```

```
[(<PIL.Image.Image image mode=L size=28x28 at 0x7F8A65A64490>, 7), (<PIL.Image.Image image mode=L size=28x28 at 0x7F8A65A64D50>, 2), (<PIL.Image.Image image mode=L size=28x28 at 0x7F8A65A64690>, 1), (<PIL.Image.Image image mode=L size=28x28 at 0x7F8A65A64810>, 0), (<PIL.Image.Image image mode=L size=28x28 at 0x7F8A65A641D0>, 4), (<PIL.Image.Image image mode=L size=28x28 at 0x7F8A65B2FBD0>, 1), (<PIL.Image.Image image mode=L size=28x28 at 0x7F8A65B2F550>, 4), (<PIL.Image.Image image mode=L size=28x28 at 0x7F8A65A64D90>, 9), (<PIL.Image.Image image mode=L size=28x28 at 0x7F8A65A64250>, 5), (<PIL.Image.Image image mode=L size=28x28 at 0x7F8A65B2F810>, 9), (<PIL.Image.Image image mode=L size=28x28 at 0x7F8A65B2F350>, 0), (<PIL.Image.Image image mode=L size=28x28 at 0x7F8A65B2FB10>, 6), (<PIL.Image.Image image mode=L size=28x28 at 0x7F8A65B2F8D0>, 9), (<PIL.Image.Image image mode=L size=28x28 at 0x7F8A65B2FD90>, 0), (<PIL.Image.Image image mode=L size=28x28 at 0x7F8A65B2F710>, 1), (<PIL.Image.Image image mode=L size=28x28 at 0x7F8A65B2F890>, 5), (<PIL.Image.Image image mode=L size=28x28 at 0x7F8A65B2F5D0>, 9), (<PIL.Image.Image image mode=L size=28x28 at 0x
```

从运行结果中可以看出，转换后的数据集对象变成了一个元组列表，每个元组有两个元素，第一个元素是图像数据，第二个元素是图像的标签。

这里图像数据是 PIL.Image.Image 类型的，这种类型可以直接在 Jupyter 中显示出来。显示一条数据的代码如下。

复制代码

```
1 display(mnist_dataset_list[0][0])
2 print("Image label is:", mnist_dataset_list[0][1])
```

运行结果如下图所示。可以看出，数据集 mnist_dataset 中的第一条数据是图片手写数字“7”，对应的标签是“7”。

```
display(mnist_dataset_list[0][0])
print("Image label is:", mnist_dataset_list[0][1])
```

```
Image label is: 7
```

好，如果你也得到了上面的运行结果，说明你的操作没问题，恭喜你成功完成了读取操作。

小结

恭喜你完成了这节课的学习。我们已经迈出了模型训练的第一步，学会了如何读取数据。

今天的重点就是掌握**两种读取数据的方法，也就是自定义和读取常用图像数据集**。

最通用的数据读取方法，就是自己定义一个 Dataset 的派生类。而读取常用的图像数据集，就可以利用 PyTorch 提供的视觉包 Torchvision。

Torchvision 库为我们读取数据提供了丰富的图像数据集的接口。我用手写数字识别这个经典例子，给你示范了如何使用 Torchvision 来读取 MNIST 数据集。

`torchvision.datasets`继承了 Dataset 类，它在预定义许多常用的数据集的同时，还预留了数据预处理与数据增强的接口。在下一节课中，我们就会接触到这些数据增强函数，并学习如何进行数据增强。

每课一练

在 PyTorch 中，我们要定义一个数据集，应该继承哪一个类呢？

欢迎你在留言区和我交流互动，也推荐你把这节课内容分享给更多的朋友、同事，跟他一起学习进步。

分享给需要的人，Ta订阅后你可得 20 元现金奖励

生成海报并分享

赞 1 提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 05 | Tensor变形记：快速掌握Tensor切分、变形等方法

下一篇 07 | Torchvision（中）：数据增强，让数据更加多样性

精选留言 (7)

写留言

Spoon
2021-10-22

老师好， 想问下。。为啥你说是10x3 维啊 数据不是2维的吗，下面标签不是一维的吗。。。有点confuse了。。
data_tensor = torch.randn(10, 3)
target_tensor = torch.randint(2, (10,)) # 标签是0或1

展开

作者回复: 你好，Spoon。感谢你的留言。
数据是3维，标签是1维，一共有10条数据。
10*3是指data_tensor的size，10*1是指target_tensor的size。
torch.randint(2, (10,)) 这句代码的意思是指生成10个随机数，随机数的范围只能是0或者1。
可以运行下代码，分别打印一下data_tensor和target_tensor的内容，一看就能明白了： ）

1 1

超人不会飞
2021-10-25

tensor_dataloader 是一个迭代器
iter (tensor_dataloader)是一个迭代器对象

iter(tensor_dataloader).next() 是输出迭代器的下一个元素

...

展开

作者回复: 你好，超人不会飞。感谢你的留言。

tensor_dataloader是一个dataloader对象，用iter()强制类型转换成迭代器的对象，next()是输出迭代器下一个元素。^^

1

Geek_273c0b

2021-10-23

老师您好。我尝试了下载MNIST数据集，然后把download设置为false，root设置为MNIST下载文件的相对路径，发现无法找到。搜了下网上资料，发现torchvision在下载之后有一系列的预处理。老师能否讲下如何直接使用原生的MNIST数据集？

展开

作者回复: 你好，Geek_273c0b。感谢你的留言。在后面实战篇中的图像分类章节中，我们会模拟一个真实的项目，包括数据处理、模型训练、模型评估等，其中数据处理中就会讲解如何读取未做任何处理的原始图片数据。

Geek_60b6fb

2021-10-23

为什么MyData中的两个method都用双前导和双末尾下划线？有什么约定吗还是编译不同？

作者回复: 你好，谢谢你的留言。

前后都有双下划线的函数是python的魔法函数。

那几个方法是在Dataset 类中定义的，

而我们继承 Dataset 类创建我们的Dataset的时候，至少需要重写那几个方法。关于前后都有双下划线函数的这块内容，你可以去看看Python有关的知识。^^

1

vcjmhg

2021-10-22

应该继承torch.utils.data.Dataset类

展开

作者回复: 你好，vcjmhg，👍👍👍👍👍。每次回答都很正确^^。加油~

IUniverse
2021-10-22

要继承Dataset类

展开

作者回复: 你好，IUniverse。感谢你留言。👍👍👍👍👍答案是正确的^^。加油~

lwg0452
2021-10-22

torch.utils.data.Dataset类

展开

作者回复: 你好，lwg0452。👍👍👍👍👍👍，每次的答案都非常正确^^。加油~